Challenge Machine Learning : Classification en phase de sommeil avec Dreem

Marion Favre d'Echallens et Jean-Louis Truong

7 janvier 2019

1. Introduction

Ce challenge est réalisé en partenariat avec l'entreprise Dreem qui est une start-up spécialisée dans l'amélioration du sommeil des personnes.

Contexte du challenge

Ce challenge consiste à réaliser de la classification en stades de sommeil. Une nuit voit en effet défiler plusieurs cycles de sommeil qui se composent tous d'une phase :

- d'éveil
- de sommeil léger
- de sommeil profond
- de sommeil paradoxal.

Un moyen de mesurer le sommeil est d'utiliser le polysomnographe qui relève notamment l'activité du cerveau, le mouvement des yeux et la tension musculaire afin d'évaluer la qualité du sommeil d'une personne.

Dans cette optique de mesure, la société Dreem a développé un bandeau qui fonctionne comme le polysomnographe et qui permet de mesurer trois types de signaux:

- l'activité électrique du cerveau grâce à un électro-encéphalogramme (EEG)
- le mouvement la position, la respiration grâce à un accéléromètre
- les battements sanguins grâce à un oxymètre de pouls.

Le challenge

Ce bandeau enregistre donc une certaine quantité de données par nuit et l'objectif de ce challenge est de développer un algorithme permettant, à partir des données de 30 secondes d'enregistrement du bandeau, dans quel stade de sommeil parmi les quatre cités plus haut se trouve la personne.

Nous avons pour cela à notre disposition 7 enregistrements d'encéphalogramme (sept positions différentes sur la tête), 1 enregistrement d'oxymètre et 3 enregistrements d'accéléromètre. Ces enregistrements sont de 30 secondes et ils sont labellisés i.e. nous connaissons le stade de sommeil associé.

2. Prétraitement des données

Les données sont présentées sous le format h5 afin de faciliter leur manipulation au vu de leur taille trés volumineuse. Nous disposons en effet de sept bases de données d'enregistrements d'encéphalogrammes contenant chacun 38289 lignes de 1500 valeurs, ce qui correspond à une fréquence de 50Hz. Les quatres autres bases de données ne contiennent que 300 valeurs par enregistrement (fréquence de 10Hz).

Afin de lire et manipuler ces données, nous utilisons le package `h5` de R.

L'objet `xtrain` contient les onze datasets à exploiter. Pour ce faire, nous les transformons en `dataframe` afin de les utiliser.

```
library(h5,warn.conflicts = FALSE)
```

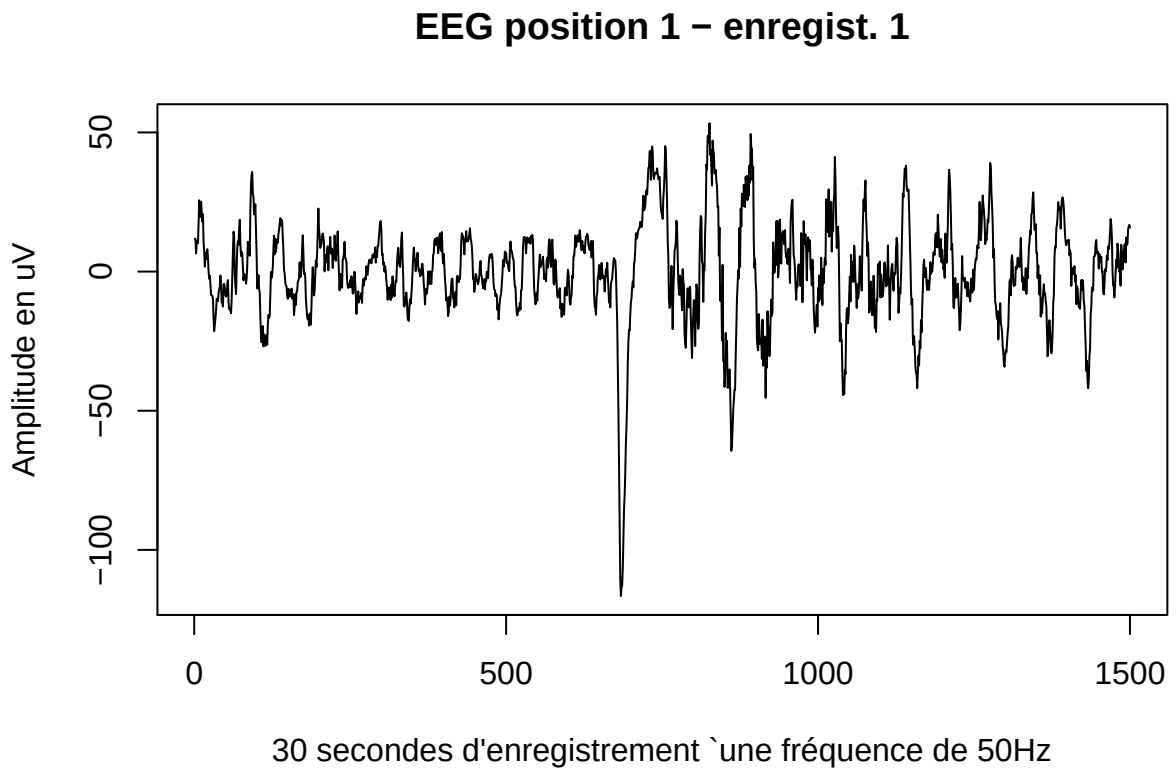
```
## Warning: package 'h5' was built under R version 3.4.4
```

```
data_folder = "C:/Users/Admin/Documents/Centrale Paris/3A/OMA/Machine Learning/Challenge/Data/"
ytrain = read.csv(paste0(data_folder,"train_y.csv"))
xtrain = h5file(name = paste0(data_folder,"train.h5/train.h5"))
list.datasets(xtrain)
```

```
##  [1] "/accelerometer_x"          "/accelerometer_y"
##  [3] "/accelerometer_z"          "/eeg_1"
##  [5] "/eeg_2"                    "/eeg_3"
##  [7] "/eeg_4"                    "/eeg_5"
##  [9] "/eeg_6"                    "/eeg_7"
## [11] "/pulse_oximeter_infrared"
```

```
eeg1 = xtrain[list.datasets(xtrain, recursive = TRUE)[4]]
eeg1 = as.data.frame(readDataSet(eeg1))
```

On peut observer le premier enregistrement EEG ci-dessous.



3. Extraction de features

Afin de construire un modèle de classification des données en stade de sommeil, nous avons extrait des signaux un certain nombre de features. Nous les avons ensuite testés en appliquant l'algorithme de classification présenté dans la section suivante afin de déterminer l'importance de leur influence sur la détermination du stade de sommeil.

Nous avons utilisé différentes approches pour le choix des features à extraire, suite à des recherches bibliographiques dont les sources sont citées plus bas.

Nous avons d'abord calculé des features basiques sur tous les signaux: l'écart-type et la moyenne du signal, la moyenne, le minimum et le maximum du signal en valeur absolue. Nous utilisons les deux fonctions suivantes pour calculer les quantités sur le signal en valeur absolue et enregistrer les données ainsi calculées.

```
## [1] "Fonction val_absolue"
```

```
## function(x)
## {
##   abx = abs(as.numeric(x))
##   ma = mean(abx)
##   max = max(abx)
##   min = min(abx)
##
##   return( c(ma,max,min))
##
## }
```

```
## [1] "Fonction calcul_feat_base2"
```

```
## function(x,train = TRUE)
## {
##   if (train)
##     df = ytrain
##   else
##     df = yrandom
##   for (i in 1:11)
##   {
##     print(i)
##     data = as.data.frame(readDataSet(x[list.datasets(x, recursive = TRUE)[i]]))
##     s = apply(data,1,val_absolue)
##     dfs = as.data.frame(t(s))
##     colnames(dfs) = c(paste0("mean_abs_",list.datasets(xtrain, recursive = TRUE)[i]),
##                       paste0("max_abs_",list.datasets(xtrain, recursive = TRUE)[i]),
##                       paste0("min_abs_",list.datasets(xtrain, recursive = TRUE)[i]))
##     df = cbind(df,dfs)
##
##   }
##     rm(data)
##     rm(dfs)
##   if (train)
##     {
##       write.csv(df,file = paste0(data_folder,"basic_abs.csv"),row.names = FALSE)
##     }
##   else
```

```
##     df = df[,3:ncol(df)]
##     write.csv(df,file = paste0(data_folder,"basic_abs_test.csv"),row.names = FALSE)
##
## }
```

Ensuite, en nous appuyant sur les ondes caractéristiques présentes dans les différents stades de sommeil, nous avons réalisé deux types de décomposition des signaux EEG. En effet, on peut distinguer quatre types d'onde entre 0 et 30Hz:

- les ondes alpha entre 8 et 13Hz
- les ondes theta entre 4 et 8Hz
- les ondes beta entre 13 et 30Hz
- les ondes delta entre 0.5 et 4Hz

Chaque stade de sommeil étant caractérisé par certains types d'onde ci-dessus, nous avons filtré le signal de chacun des 7 enregistrements EEG entre 0 et 30Hz et decoupé en quatre plages correspondant aux ondes ci-dessus. Sur chacune des plages, nous avons calculé la moyenne, la somme des amplitudes en valeur absolue ainsi que le ratio de la somme des amplitudes en valeur absolue de la plage sur celle du signal total filtré, que l'on peut appeler amplitude relative. Nous calculons également la somme des amplitudes au carré du signal total ainsi que les proportions de chacune des plages de fréquences dans le signal. Les fonctions que nous utilisons pour calculer ces features sont les suivantes et font appel à la bibliothèque seewave:

```
## Warning: package 'seewave' was built under R version 3.4.4
```

```
## [1] "Fonction freq_prop: proportions des frequences alpha, theta, beta, delta pour EEG"
```

```
## function(x)
## {
##   x_filtered = ffilter(as.numeric(x),f=50,from = 0, to = 30)
##   pic = as.data.frame(fpeaks(seewave::spec(as.numeric(x_filtered),f = 50,plot = FALSE),plot = FALSE)
##   pic$freq = pic$freq*1000
##   n =  nrow(pic)
##   #alpha waves
##   f_alpha =  subset(pic,pic$freq >= 8 & pic$freq <= 13)
##   alpha = nrow(f_alpha)/n
##
##   #theta waves
##   f_theta =  subset(pic,pic$freq >= 4 & pic$freq <= 8)
##   theta = nrow(f_theta)/n
##
##   #delta waves
##   f_delta =  subset(pic,pic$freq >= 0.5 & pic$freq <= 4)
##   delta = nrow(f_delta)/n
##
##   #beta waves
##   f_beta =  subset(pic,pic$freq >= 13 & pic$freq <= 30)
##   beta = nrow(f_beta)/n
##
##   return(c(alpha,beta,delta,theta))
## }
```

```
## [1] "Fonction freq_feat: features des frequences alpha, theta, beta, delta pour EEG"
```

```
## function(x)
## {
##   x_filtered = ffilter(as.numeric(x),f=50,from = 0, to = 30)
##   pic = as.data.frame(fpeaks(seewave::spec(as.numeric(x_filtered),f = 50,plot = FALSE),plot = FALSE)
##   pic$freq = pic$freq*1000
##   #all
##   amp = sum(abs(pic$amp))
##   amp2 = sum(pic$amp^2)
##   m = mean(pic$freq)
##   s = sd(pic$freq)
##
##   #alpha waves
##   f_alpha =  subset(pic,pic$freq >= 8 & pic$freq <= 13)
##   alpha_amp = sum(abs(f_alpha$amp) )
##   alpha_amp_rel = alpha_amp/amp
##   alpha_m = mean(f_alpha$freq)
##
##   #theta waves
##   f_theta =  subset(pic,pic$freq >= 4 & pic$freq <= 8)
##   theta_amp = sum(abs(f_theta$amp) )
##   theta_amp_rel = theta_amp/amp
##   theta_m = mean(f_theta$freq)
##
##   #delta waves
##   f_delta =  subset(pic,pic$freq >= 0.5 & pic$freq <= 4)
##   delta_amp = sum(abs(f_delta$amp) )
##   delta_amp_rel = delta_amp/amp
##   delta_m = mean(f_delta$freq)
##
##   #beta waves
##   f_beta =  subset(pic,pic$freq >= 13 & pic$freq <= 30)
##   beta_amp = sum(abs(f_beta$amp) )
##   beta_amp_rel = beta_amp/amp
##   beta_m = mean(f_beta$freq)
##
##   return(c(amp,amp2,m,s,
##            alpha_amp,alpha_amp_rel,alpha_m,
##            theta_amp,theta_amp_rel,theta_m,
##            delta_amp,delta_amp_rel,delta_m,
##            beta_amp,beta_amp_rel,beta_m))
##
##
## }
```

Parallèlement à cette décomposition, nous réalisons une décomposition en quatre ondelettes des enregistrements EEG. Cette décomposition se fait après un filtre de Daubechies appliqué au signal. On obtient alors les coefficients de chacune des 4 ondelettes. Nous calculons ensuite l'écart-type et l'entropie de Renyi de chaque ondelette. L'entropie est une mesure de l'énergie du signal, nous la calculons par une fonction de la bibliothèque seewave de R avec un coefficient de 0.5. Les calculs se font comme ceci grâce à la bibliothèque wavelets:

```
## Warning: package 'wavelets' was built under R version 3.4.4
```

```
## [1] "Fonction wavelets_coeff4: features sur les ondelettes pour EEG"
```

```
## function(x)
## {
##   d = dwt(as.numeric(x),n.levels = 4,filter = "d20")
##
##   wave1_ent = sh(spec(nonvide(d@W$W1),f=50,plot = FALSE), alpha =0.5)
##   wave1_sd = sd(nonvide(d@W$W1))
##
##   wave2_ent = sh(spec(nonvide(d@W$W2),f=50,plot = FALSE), alpha =0.5)
##   wave2_sd = sd(nonvide(d@W$W2))
##
##   wave3_ent = sh(spec(nonvide(d@W$W3),f=50,plot = FALSE), alpha =0.5)
##   wave3_sd = sd(nonvide(d@W$W3))
##
##   wave4_ent = sh(spec(nonvide(d@W$W4),f=50,plot = FALSE), alpha =0.5)
##   wave4_sd = sd(nonvide(d@W$W4))
##
##   return(c(wave1_ent,wave1_sd,wave2_ent,wave2_sd,wave3_ent,
##            wave3_sd,wave4_ent,wave4_sd))
## }
```

Après ces premiers calculs de features, nous remarquons que le stade de sommeil qui semble le plus difficile à classer est le stade 1. Ce dernier est caractérisé notamment par les ondes alpha, c'est pourquoi nous décidons de calculer des features supplémentaires uniquement sur la plage de fréquence alpha des signaux EEG. Nous calculons ainsi l'écart-type, l'entropie de Renyi et la min-max distance de cette plage. La min-max distance est la somme des distances entre les points maximum et minimal de chaque intervalle de temps du signal (découpé en n intervalles). Les fonctions sont les suivantes :

```
## [1] "Fonction MMD: min_max distance"
```

```
## function(x){
##   x = as.numeric(x)
##   if (length(x) > 100)
##     lambda = 100
##   if (length(x) > 10 && length(x) <= 100)
##     lambda = 10
##
##   res = 0
##   n = length(x)/lambda
##
##   if (n > 10)
##   {
##     for (i in 1:n)
##     {
##       x_i=x[(1 + (i-1)*lambda) : (i*lambda)]
##       My = max(x_i)
##       Mx = seq(along = x_i)[x_i== My]
##       my = min(x_i)
##       mx = seq(along = x_i)[x_i== my]
##
##       res = res + sqrt((mx - Mx)^2 + (my - My)^2)
##     }
##   }
##
```

```
##   return(res)
## }
```

```
## [1] "Fonction alpha: features sur frequences alpha pour EEG"
```

```
## function(x)
## {
##   x_alpha = ffilter(as.numeric(x),f=50,from = 8, to = 13)
##   ent = sh(spec(nonvide(x_alpha),f=50,plot = FALSE), alpha =0.5)
##   mmd = MMD(nonvide(x_alpha))
##   sd = sd(nonvide(x_alpha))
##   return(c(ent,mmd,sd))
## }
```

La fonction nonvide est introduite pour s'affranchir des listes vides éventuelles et les remplacer par une liste contenant un zéro:

```
## function(x){
##   if (length(x) == 0)
##     return(0)
##   else
##     return(x)
## }
```

À chaque fois, nous calculons les features sur chaque enregistrement de EEG d'apprentissage et de test puis nous regroupons tous les dataframes en un seul au moment de la création du modèle. Ci-dessous un exemple pour les features sur les plages d'ondes alpha:

```
## [1] "Fonction calcul_feat_alpha: calcul des features sur ondes alpha"
```

```
## function(x,train = TRUE) #x = xtrain ou xtest
## {
##   for (i in 4:10)
##   {
##     print(i)
##     data = as.data.frame(readDataSet(x[list.datasets(x, recursive = TRUE)[i]]))
##     s = apply(data,1,alpha)
##     df = as.data.frame(t(s))
##     colnames(df) = c(paste0("alpha_ent",i-3),
##                      paste0("alpha_mmd",i-3),
##                      paste0("alpha_sd",i-3))
##
##     if (train)
##     {
##       df =cbind(ytrain,df)
##       write.csv(df,file = paste0(data_folder,"alpha_eeg",i-3,".csv"),row.names = FALSE)
##     }
##     else
##       write.csv(df,file = paste0(data_folder,"alpha_egg",i-3,"_test.csv"),row.names = FALSE)
##   }
##   rm(data)
## }
```

```
## [1] "Fonction rassembler_feat_alpha: regrouper les features sur ondes alpha"

## function(train = TRUE)
## {
##   if (train)
##   {
##     df = read.csv(paste0(data_folder,"alpha_eeg1.csv"))
##     for (i in 2:7)
##     {
##      data = read.csv(paste0(data_folder,"alpha_eeg",i,".csv"))
##
##       df = merge(df,data,by=c("id","sleep_stage"),all.x = TRUE,all.y = TRUE)
##     }
##     df$sleep_stage = as.factor(df$sleep_stage)
##   }
##
##   else
##   {
##     df = read.csv(paste0(data_folder,"alpha_egg1_test.csv"))
##     for (i in 2:7)
##     {
##       data = read.csv(paste0(data_folder,"alpha_egg",i,"_test.csv"))
##       df =cbind(df,data)
##     }
##   }
##   rm(data)
##   return(df)
## }
```

4. Modèle utilisé - Description théorique

Nous utilisons l'algorithme de Random Forest pour la classification. Il s'agit d'un algorithme d'agrégation de modèles (boosting) qui consiste à rassembler des modèles simples afin d'aboutir à un modèle final performant. L'algorithme de RandomForest consiste en effet à générer un certain nombre de modèles simples (arbres de décision). Chaque modèle est généré à partir d'un échantillon bootstrap des données, c'est-à-dire que l'on pioche avec remise n individus parmi N pour construire l'échantillon. On choisit également aléatoirement un sous-ensemble de variables pour créer le modèle sur cet échantillon. Il y a ainsi deux niveaux d'aléatoire dans l'algorithme de RandomForest: au niveau des individus et des variables piochés pour chaque modèle simple.

L'algorithme permet d'évaluer des variables les plus influentes dans le modèle et donc de créer ensuite des modèles contenant moins de variables (celles jugées comme plus importantes). Ceci nous a donc permis de sélectionner les principaux features de notre modèle et obtenir ainsi un modèle plus performant que celui contenant l'ensemble des variables.

Les features sélectionnés pour notre meilleur modèle (meilleur score) sont:

```
#"beta_amp_rel4", amplitude relative pour les ondes beta sur l'EEG 4;
#"alpha_sd2", ecart type du signal des ondes alpha sur l'EEG 2;
#"max_abs_.accelerometer_y", maximum de la valeur absolue du signal de l'accéléromètre suivant y;
#"mean_abs_.accelerometer_y", moyenne de la valeur absolue du signal de l'accéléromètre suivant y;
#"mean_abs_.eeg_4", moyenne de la valeur absolue du signal de l'EEG 4;
#"wave2_sd_eeg2", écart-type des coefficients de l'ondelette 2 de l'EEG 2;
#"mean_abs_.accelerometer_x", moyenne de la valeur absolue du signal de l'accéléromètre suivant x;
```

```
#"mean_abs_.accelerometer_z", moyenne de la valeur absolue du signal de l'accéléromètre suivant z;
#"beta_amp_rel6", amplitude relative pour les ondes beta sur l'EEG 6;
#"alpha_amp_rel2", amplitude relative pour les ondes alpha sur l'EEG 2;
#"beta_amp_rel2", amplitude relative pour les ondes beta sur l'EEG 2;
#"max_abs_.accelerometer_x", maximum de la valeur absolue du signal de l'accéléromètre suivant x;
#"wave3_sd_eeg2", écart-type des coefficients de l'ondelette 3 de l'EEG 2;
#"mean_abs_.eeg_2", moyenne de la valeur absolue du signal de l'EEG 2;
#"delta_amp_rel4", amplitude relative pour les ondes delta sur l'EEG 4;
#"mean_abs_.eeg_6", moyenne de la valeur absolue du signal de l'EEG 6;
#"mean_abs_.pulse_oximeter_infrared", moyenne de la valeur absolue du signal de l'oxymètre;
#"mean_abs_.eeg_5", moyenne de la valeur absolue du signal de l'EEG 5;
#"wave4_sd_eeg4", écart-type des coefficients de l'ondelette 4 de l'EEG 4;
#"mean_abs_.eeg_7", moyenne de la valeur absolue du signal de l'EEG 7;
#"wave3_sd_eeg4", écart-type des coefficients de l'ondelette 3 de l'EEG 4;
#"max_abs_.eeg_2", maximum de la valeur absolue du signal de l'EEG 2;
#"wave2_ent_eeg4", entropie des coefficients de l'ondelette 2 de l'EEG 4;
#"max_abs_.pulse_oximeter_infrared", maximum de la valeur absolue du signal de l'oxymètre;
#"min_abs_.accelerometer_x", minimum de la valeur absolue du signal de l'accéléromètre suivant x;
#"wave1_ent_eeg2", entropie des coefficients de l'ondelette 1 de l'EEG 2;
#"max_abs_.accelerometer_z", maximum de la valeur absolue du signal de l'accéléromètre suivant z;
#"wave4_sd_eeg2", écart-type des coefficients de l'ondelette 4 de l'EEG 2;
#*"min_abs_.accelerometer_y", minimum de la valeur absolue du signal de l'accéléromètre suivant y;
#"wave1_sd_eeg2", écart-type des coefficients de l'ondelette 1 de l'EEG 2.
```

Nous créons le modèle avec 700 arbres (ntree) et 48 variables piochées à chaque sous-modèle fabriqué (mtry). La commande est la suivante:

```
# fichier contenant les variables
# imp = read.csv(paste0(data_folder,"imp2.csv"))  d'importance
# imp[,1] = as.character(imp[,1])
#
# f_RandomForest3 = randomForest(sleep_stage~.,
#                                data=df[,c("sleep_stage",subset(imp,imp[,2] > 100)[,1])],
# ntree=700,mtry = 48)
```

Remarque : Il y a 71 variables dans le modèle sur plus de 200 calculées et testées, ce qui reste beaucoup. Ce nombre de variables est cependant le compromis que l'on a trouvé pour obtenir un bon score.

Vous trouverez dans les trois fichiers .R joints le code complet utilisé pour la génération du modèle:

- fonctions_final.R: code contenant les fonctions pour calculer les features.
- features_final.R: code contenant les calculs des features et leur enregistrement dans des tableaux csv.
- source_final.R: code appelant les deux fichiers précédents, générant les features et le tableau final des features, créeant le modèle final et calculant enfin les classes de l'échantillon test.

5. Protocole de validation croisée

La validation croisée est réalisée par l'algorithme de Random Forest lui-même (le "Out-of-the-bag error estimate"). Pour chaque arbre simulé (i.e. pour chaque sous-modèle simple créé), deux tiers des données tiré aléatoirement est utilisé pour l'apprentissage et le reste pour le test du modèle. À chaque simulation, l'algorithme simule l'arbre de classification avec les données d'apprentissage, puis classe les données test à partir de cet arbre.

Après la simulation de tous les arbres, l'algorithme calcule la "out-of-the-bag error estimate" comme suit : pour chaque individu "n" out-of-the-bag d'un sous-modèle, l'algorithme lui attribue la classe "j" comme étant celle pour laquelle il a été classé le plus grand nombre de fois. Ainsi, la "out-of-the-bag error" de l'individu "n" est obtenue en prenant le nombre de fois où "j" n'est pas la bonne classe de l'individu divisé par le nombre total d'individu. La OOB error affichée est alors la moyenne des erreurs sur tous les individus out-of-the-bag de tous les sous-modèles.

Cette erreur calculée est donc de la validation croisée car les individus out-of-the-bag constituent l'échantillon test avec en moyenne un tiers des données (voir graphique de la section suivante.)

6. Présentation des résultats

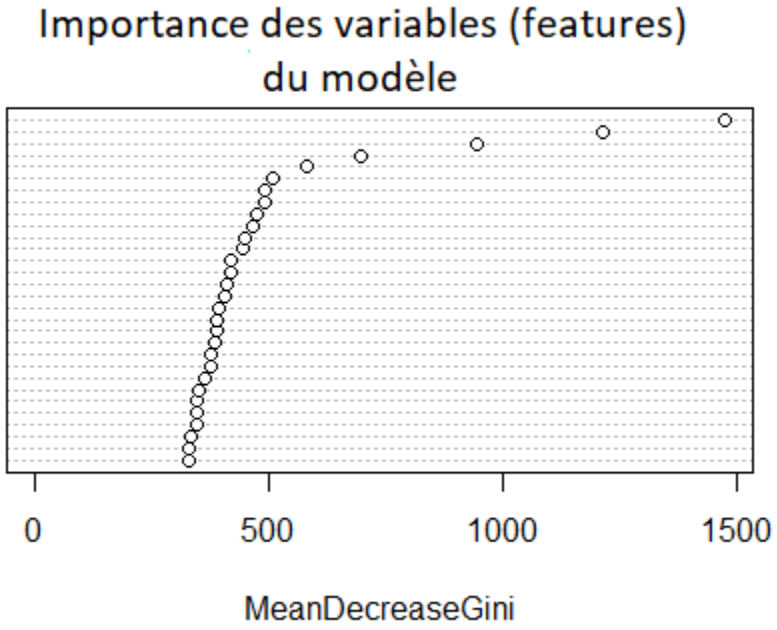
Nous avons choisi 700 arbres pour notre algorithme de "random forest". Ce nombre semblait être un bon compromis entre la fiabilité de nos résultats et le temps de calcul.

Pour le nombre de variables tirées aléatoirement avec remise à chaque étape de l'algorithme, nous avons choisi 48. En effet, étant donné le nombre important de variables de notre matrice (plus de 200), il est logique d'augmenter également le nombre de variables piochées aléatoirement lors de la construction de l'échantillon boostraop pour chaque sous-modèle. Plus ce nombre augmentait, plus notre modèle était satisfaisant, c'est pourquoi nous choisissons 48 qui semblait être le nombre maximal possible dans la fonction.

La matrice de transition du modèle final sur le training set avec la validation croisée (erreur out-of-the-bag) réalisée par la fonction RandomForest de R est la suivante:

```
# Call:
#  randomForest(formula = sleep_stage ~ ., data = df[, c("sleep_stage",
#subset(imp, imp[, 2] > 100)[, 1])], ntree = 700, mtry = 48)
#                Type of random forest: classification
#                      Number of trees: 700
# No. of variables tried at each split: 48
#
#         OOB estimate of  error rate: 26.33%
# Confusion matrix:
#      0  1     2    3    4 class.error
# 0 2460  6   753   18  382   0.3202542
# 1  313 28   667   12  333   0.9793052
# 2  371  4 14655  566 1536   0.1445832
# 3   97  0  1564 4021   49   0.2983772
# 4  292  8  3065   44 7045   0.3260953
```

Le taux d'erreur est de 26,33% avec un stade 1 qui a le plus mauvais taux de classement correct malgré les features sur les ondes alpha censés le caractériser davantage. Le stade 2 est le mieux classé et les trois autres stades ont des taux d'erreur similaires. On remarque également que la majorité des erreurs de classement se fait en classant le stade dans le stade 2 au lieu de son stade réel.

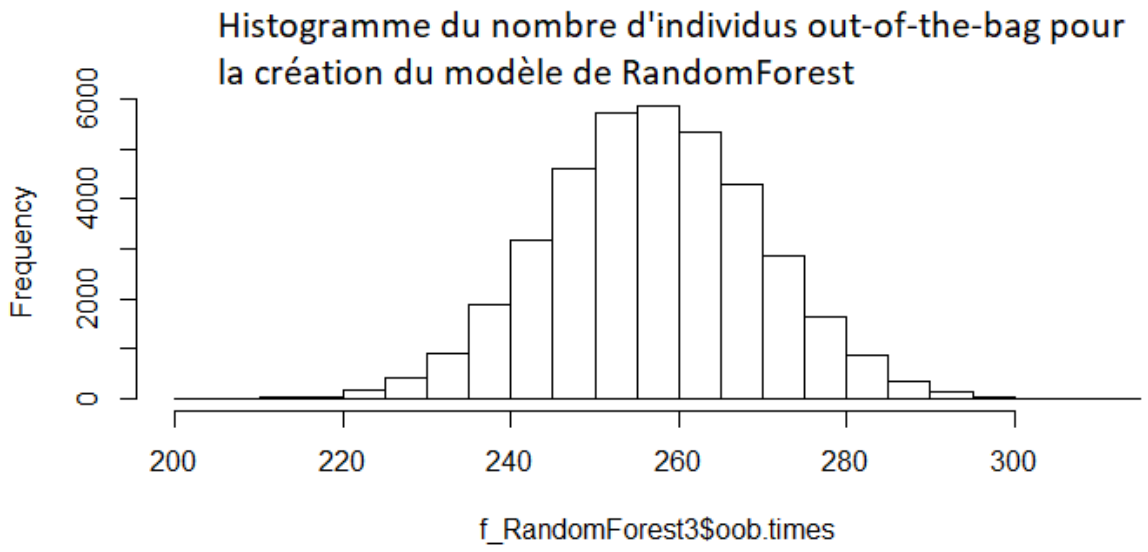


Sur ce graphe nous avons présenté l'importance des variables du modèle et leur score associé. Un point correspond à une variable. Nous voyons donc que les cinq premières variables les plus importantes se démarquent beaucoup des autres en ayant un score largement supérieur aux autres, les autres ayant un score décroissant mais relativement proche les unes des autres. Nos tests montrent cependant que les autres variables sont essentielles pour le modèle. Les plus importantes sont :

- Amplitude relative pour les ondes beta sur l'EEG 4
- Ecart type du signal des ondes alpha sur l'EEG 2
- Maximum de la valeur absolue du signal de l'accéléromètre suivant y
- Moyenne de la valeur absolue du signal de l'accéléromètre suivant y

- Moyenne de la valeur absolue du signal de l'EEG 4

On remarque ici que les enregistrements EEG 2 et 4 et de l'accéléromètre suivant y semblent être les plus influents dans la classification.



Ce graphe présente l'histogramme du nombre de fois où une observation est out-of-the-bag, c'est-à-dire non piochée lors de la création de l'échantillon bootstrap (données d'apprentissage) pour un sous-modèle et utilisé comme données test pour le sous modèle considéré. On voit donc que la sélection aléatoire des données "out-of-the-bag" suit une gaussienne, de moyenne environ 255, ce qui correspond à un peu plus du tiers du nombre d'arbres, c'est-à-dire de sous-modèles fabriqués pour le modèle final. Ce résultat est cohérent puisque que l'on sépare en théorie nos données en 2/3 apprentissage et 1/3 test dans chaque arbre simulé. Notre simulation est donc cohérente.

7. Bibliographie

- Reza Boostani, Foroozan Karimzadeh, Mohammad Torabi-Nami. A Comparative Review on Sleep Stage Classification Methods in Patients and healthy Individuals. 2016.
- Khald Ali I. Aboalayon, Miad Faezipour, Wafaa S. Almuhammadi and Saeid Moslehpour.Sleep Stage Classification Using EEG Signal Analysis: A Comprehensive Survey and New Investigation
- RUEY-SONG HUANG, LING-LING TSAI, AND CHUNG J. KUO. Selection of Valid and Reliable EEG Features for Predicting Auditory and Visual Alertness Levels
- Leo Breiman and Adele Cutler, Random Forests, Cours de l'université de Berkeley, Californie